Performance Characterization of Decentralized Algorithms for Replica Selection in Distributed Object Systems*

Ceryen Tan
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139
1-617-253-1000
ctan@mit.edu

Kevin Mills
National Institute of Standards and Technology
Gaithersburg, Maryland 20899
1-301-975-3618
kmills@nist.gov

ABSTRACT
Designers of distributed systems often rely on replicas for increased robustness, scalability, and performance. Replicated server architectures require some technique to select a target replica for each client transaction. In this paper, we use simulation to characterize performance (response time, selection error, probability of server overload) for four common replica-selection algorithms (random, greedy, partitioned, weighted) when applied in a decentralized form to client queries in a distributed object system deployed on a local network. We introduce two new selection algorithms (balanced and balanced-partitioned) that give improved performance over the more common algorithms. We find the weighted algorithm performs best among the common algorithms and the balanced algorithm performs best among all those we considered. Our findings should help designers of distributed object systems to make informed decisions when choosing among available replica-selection algorithms.

Categories and Subject Descriptors
D.1.3 [**Concurrent Programming**]: Distributed Programming

General Terms
Algorithms, Design, Measurement, Performance

Keywords
Distributed Object Systems, Replica Selection.

1. INTRODUCTION
Designers of distributed systems often rely on replicas for increased robustness, scalability, and performance. Replication appears in a growing range of applications, such as web services [1-15], distributed object systems [16-20], grid systems [21-22], and content distribution networks [25-26]. Replication systems require that each client transaction be assigned to a specific server replica for processing. Selection (or assignment) algorithms aim to minimize client response time, to balance server load, or to achieve a combination. Typical commercial systems for server replication [10-15] allow a designer to choose among several alternate selection algorithms; however, the designer is given little quantitative information to aid in choosing. At best, commercial systems outline heuristics to differentiate among available algorithms. Even academic papers [e.g., 1-9] do not give comprehensive quantitative results.

In this paper, we aim to help designers understand quantitative performance differences (and underlying causes) among the most common algorithms (*random*, *greedy*, *partitioned*, and *weighted*) for replica selection. We also introduce two new algorithms (*balanced* and *balanced-partitioned*), and compare performance with the more common algorithms. We consider three performance characteristics: average client response time, probability of selection error, and probability of server overload.

Section 2 surveys common selection algorithms typically implemented in commercial systems and identifies some algorithms proposed by researchers. Section 3 explains the design of our experiment, including performance metrics. Section 4 presents simulation results, which are discussed in Section 5. We conclude in Section 6.

2. REPLICA SELECTION
Our literature survey revealed two classes of replica-selection algorithms. One class encompasses heuristically based, statically configured algorithms. One static algorithm uses a *round robin* approach [10,13,15] to rotate client transactions in turn among replicas. A similar algorithm (using a uniform distribution) *randomly assigns* [10,11,13] each client transaction to one of the available replicas. These two algorithms assume that each replica has similar processing power available and that the mix of transaction types is congruent among the client population. Absent these assumptions, the round robin and random algorithms could perform poorly; however, no dynamic measurements are needed for either algorithm. A third approach uses a *proportional* algorithm [13,14], which distributes client transactions among replicas in proportion to relative power ratings assigned by a system administrator. This accounts for variation in processing power when a server population consists of heterogeneous platforms. Here, some information must be collected (off-line) and encoded for use by the algorithm, which cannot adapt should configuration information prove inaccurate or transient. Our experiments investigate algorithms that dynamically adjust assignment of client transactions based on measured conditions; thus, we do not consider statically configured approaches. We do simulate random assignment as a baseline case.

A second class of selection algorithms dynamically assigns client transactions based on measured conditions. The most common approach, *greedy* selection, [3,7,9-11,14,18, 23-25] assigns each transaction to the replica estimated to give best performance

WOSP'05, July 12–14, 2005, Palma, de Mallorca, Spain.
ACM 1-59593-087-6/05/0007

against some metric (different systems adopt different metrics). Greedy selection exhibits a well-known undesirable behavior where transactions oscillate in groups among available replicas. To combat this "thundering herd" effect, some systems incorporate a *weighted* algorithm [1,8,9,11,15] that first estimates the performance of each replica against a selected metric and then distributes client transactions in proportion to the likelihood that each replica will provide acceptable performance. Some systems first *partition* [1,2,5,15-17,20] replicas (based on estimated performance against some metric) into two groups, available and unavailable, and then, using greedy [2,15,20], weighted [1], random [5,16], or multicast [17] selection, assign client transactions among replicas in the available group. Multicast selection sends a transaction to every replica in the available set and uses the first returned result. Our experiment investigates the performance of three, common dynamic replica-selection algorithms: greedy, weighted, and partitioned (with random assignment).

Most replica-selection systems that we examined adopt a selection metric from one of two classes: client response time or server load. Estimated response time, an ideal selection metric from the client perspective, can be decomposed [17] as the sum of communications delay (C_D), server queuing delay (S_Q), and server processing time (S_P). C_D is important when clients access replicas through the Internet. S_Q is salient when a server is heavily loaded. S_P can dominate when transactions are computationally intensive. From a server perspective, estimated server load is an ideal selection metric. An alternative is estimated server latency (S_L), which can be decomposed as $S_Q + S_P$, yielding a convenient relationship between response time and server load. When C_D is similar among all clients, S_L provides a reasonable approximation of relative response time. When highly variable, C_D should be measured independently. Our experiments use S_L as an estimator for client response times because we simulate a distributed object system deployed on a local network, where clients experience similar communication delays.

3. EXPERIMENT DESIGN

We designed an experiment to meet the following objective: Given a set of *r* replicas deployed in a local network and queried periodically by *c* clients, characterize and compare performance of alternate selection algorithms. Our experiments exhibit the following constraints: (1) client-director pairs are deployed in a decentralized architecture (see Figure 1), (2) replicas are implemented as Jini lookup services, (3) each replica executes on a distinct, but similar, server, (4) each server is shared with other applications, and (5) replica state is piggybacked on existing Jini multicast announcements. Below, we provide details about the experiment architecture, key parameters, our technique to vary processor availability, selection metric and algorithms, and performance metrics.

3.1 Experiment Architecture

Figure 1 outlines the experiment architecture, which implements five replicas, each simulating a Jini [27] lookup service and a set of unrelated applications. Using Jini discovery and registration procedures all Jini services (not shown) register a service description with each replica. Each client periodically queries its local director (that uses some selection algorithm) to determine the address of a replica, and then queries the selected replica for service descriptions. Each client query is initiated 30 s after receiving a reply to the previous query (the first query is issued after a random startup delay). Each replica periodically (every 60 s) multicasts a Jini announcement extended to include two elements of replica state: (1) the number (*N*) of pending queries and (2) the current query processing rate (*Q*).

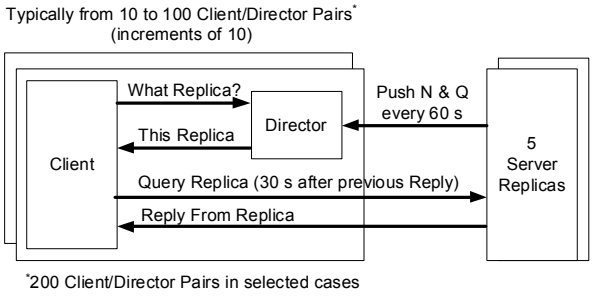

Figure 1. Experiment Architecture

Table 1. Key Experiment Parameters

Component Quantities	Servers	5	Per run constant
	Directors	10 to 100* in increments of 10 (*200 in selected cases)	Clients and Directors are paired
	Clients		
Component Startup Delays	Servers	0…15 s	Randomly selected using a uniform distribution
	Directors		
	Clients	60..75 s	
Component Workloads	Servers	Background Load is 25% to 99% (i.e., capacity for queries is 75% to 1%)	Per Server - varied every 60 s
	Directors	5 updates per minute	Each Server pushes an update every 60 s, but startup delays cause update offsets
	Clients	Client issues next query 30 s after receiving reply from previous query	Startup delays cause offset in Client queries

3.2 Key Experiment Parameters

Table 1 summarizes key parameters in three classes: component quantities, startup delays, and workloads. In most instances, an experiment considers an increasing population of clients from 10 to 100 (in increments of 10); however, the balanced and balanced-partitioned algorithms require 200 clients to distinguish their performance. The (uniformly distributed) random startup delays for servers and directors are required by Jini, while higher startup delay for clients allows Jini discovery and registration to complete before initiating client queries. Each server reserves a minimum of 1% of its processing capacity for client queries; however, as much as 75% may be used for client queries, depending upon the server's background load, which we vary every 60 s.

3.3 Processor Availability

Table 2 exhibits parameters controlling variation in processor availability. Each server reserves a minimum (BL_{MIN}) and maximum (BL_{MAX}) percentage (25% to 99%) of its capacity to process a background workload, which also defines a maximum (C_{MAX}) and minimum (C_{MIN}) capacity (75% to 1%) each server can devote to processing client queries. An unloaded server can process Q_{RATE} = 4 queries/s (assuming a query can be processed in QP_{TIME} = 250 ms), which means that a loaded server's query processing rate may vary from a minimum (Q_{MIN}) of 0.04 queries/s to a maximum (Q_{MAX}) of 3 queries/s.

Table 2. Parameters Controlling Query Processing Rate

	Parameter	Value	Explanation
Bounds on Server Background Load	BL_{MIN}	0.25	A minimum of 25% of each Server is reserved for processing a background workload
	BL_{MAX}	0.99	Up to 99% of each Server may be allocated to process the background workload
Bounds on Server Query Capacity	C_{MAX}	0.75	1 - BL_{MIN} defines the maximum % of each Server that can be allocated to process Client queries
	C_{MIN}	0.01	1 - BL_{MAX} defines the minimum % of each Server that can be allocated to process Client queries
	QP_{TIME}	250 ms	Time to process a single query on an unloaded Server
	Q_{RATE}	4 queries/s	1/ QP_{TIME} defines the rate (in queries per second) at which an unloaded Server can process queries
	Q_{MAX}	3 queries/s	$Q_{RATE} \cdot C_{MAX}$ defines the rate at which a minimally loaded Server can process queries
	Q_{MIN}	0.04 queries/s	$Q_{RATE} \cdot C_{MIN}$ defines the rate at which a maximally loaded Server can process queries
Variation in Server Query Capacity	C_D	-20	The maximum % that a Server's query capacity can decrease between updates
	C_I	+20	The maximum % that a Server's query capacity can increase between updates
	dC	−0.2 to +0.2	Selected every 60 s from a discrete uniform distribution, dC = discrete_uniform(C_D, C_I) · 0.01
	C_t	$C_{t-1} + dC$	Computed every 60 s from new dC and previous C, but constrained as follows: $C_{MIN} \le C_t \le C_{MAX}$
	Q_t	$Q_{RATE} * C_t$	Computed every 60 s, after selecting dC and computing C_t (note that $Q_{MIN} \le Q_t \le Q_{MAX}$)

Every 60 s each server updates capacity (C_t) for processing queries, subject to a constraint that capacity may not change by more than 20% (C_D and C_I bound the maximum percentage of decrease and increase, respectively) from the previous capacity (C_{t-1}). The updated capacity determines the current query-processing rate (Q_t). Figure 2 displays a two-hour time series depicting the relationship between changes in available capacity (C_t) – left-hand y-axis and query-processing rate (Q_t) – right-hand y-axis.

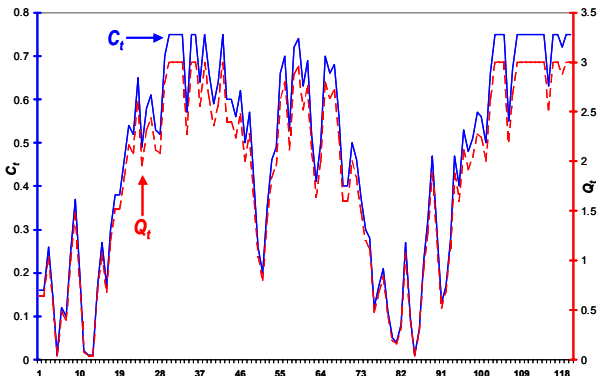

Figure 2. Variations in C_t causing Variations in Q_t

We assume each query requires similar processing, i.e., transactions are homogeneous. We also assume query-processing rate remains stable between announcements because the schedulers in the server operating systems allocate portions of processor time to specific processes and periodically (each minute here) adjust that allocation. We further assume that communication delays will be insignificant (and similar) because we simulate deployment in a local network.

3.4 Selection Metric and Algorithms

Directors select replicas based on estimated latency for each server r (S_{Lr}). $S_{Lr} = N_r/Q_r$, where N_r and Q_r are the number of queries pending and the query processing rate, respectively, received in the most recent announcement from server r. Table 3 defines key elements of the notation we use in the following description of our replica-selection algorithms.

Table 3. Notation for Defining Selection Algorithms

Notation	Explanation
S	Set of Servers
n	Number of Servers
s_i	ith Server
N_i	Number of queries backlogged at Server i
Q_i	Number of queries/second that Server i can process
T_{QAVAIL}	Server i is available when $N_i / Q_i \le T_{QAVAIL}$
A	Set of available Servers
a	Number of available Servers
W	Set of Servers with weights
w_i	Weight assigned to ith Server
K	Normalization factor
T_{QREF}	Maximum estimated Server latency
D	Set of Servers with additional queries needed to match T_{QREF}
d_i	Number of additional queries needed for ith Server to match T_{QREF}

Our baseline algorithm is *random selection*, where a director selects one of the known replicas, with each replica having an equal selection probability. Let S be the set of known server replicas, and n be the number of known server replicas. The director selects server replica s_i, where i is an integer selected uniformly on the interval [1..n]. The *greedy* algorithm requires a director to select the replica s_i with the lowest estimated server latency (N_i/Q_i).

In the *partitioned* (random selection) algorithm, a director first uses the state information cached for each replica to subset S into a set (A) of a available replicas with estimated server latencies at or below a threshold (T_{QAVAIL}). The director then selects one replica (randomly) from A. If no replicas qualify for set A, then the director selects a replica randomly from set S.

In the *weighted* algorithm, a director assigns each replica a weight based upon the inverse of estimated server latency and apportions the unit interval according to the weights. The director then draws a random real number uniformly distributed on the unit interval and selects the replica assigned to the corresponding portion.

The greedy, partitioned, and weighted algorithms consider estimated server latency (N/Q) as a unified metric; however, replicas with similar server latency estimates could possess different capacities to absorb work. This observation led us to devise a *balanced* algorithm, where a director assigns each replica a weight, based on the number of queries (d_i) required for its server latency to reach the maximum estimated server latency among all replicas. The director then apportions the unit interval according to those weights and chooses a random real number uniformly distributed on the unit interval, selecting the replica assigned to the corresponding portion. Balanced selection entails some probability that client transactions may be assigned to overloaded replicas. For this reason, we devised a *balanced-partitioned* variant, which first partitions replicas into two subsets, available and unavailable, based on comparing estimated server latency against T_{QAVAIL}, and then uses the balanced algorithm to select a replica from among the available subset. Where the available subset is empty, selection is made using the balanced algorithm.

3.5 Performance Metrics

To compare performance among selection algorithms, we define three metrics: average client response time (avg_{RT}), defined in Table 4, and probability of selection error ($prob_{SE}$) and server overload ($prob_{SO}$), defined in Table 5.

Table 4. Definition of Average Client Response Time

Notation	Definition
c	Number of Clients
q_i	Number of queries sent by Client i
$tq_{i,j}$	Time that Client i issued its jth query
$tr_{i,j}$	Time that Client i received a reply to its jth query
avg_{RT}	Average response time, computed as: $(\sum_{i=1}^{c}\sum_{j=1}^{q_i} tr_{i,j} - tq_{i,j}) / \sum_{k=1}^{c} q_k$

Table 5. Definition of Probability of Selection Error and Probability of Server Overload

Notation	Definition
s	Number of Servers
N_i	Number of queries backlogged at Server i
Q_i	Number of queries/second that Server i can process
T_{Qi}	Estimated time for Server i to clear its query backlog, computed as N_i/Q_i
T_{QMAX}	Server i is overloaded when $T_{Qi} > T_{QMAX}$ (here $T_{QMAX} = 50$ s)
T_{Oi}	Total time during which $T_{Qi} > T_{QMAX}$ for Server i (observed and updated upon arrival and departure of each query)
T_{Ui}	Total time during which Server i is up
N_{Oi}	Number of queries received by Server i when $T_{Qi} > T_{QMAX}$
N_{Ti}	Total number of queries received by Server i
$prob_{SO}$	Probability a Server is overloaded, computed as: $\sum_{i=1}^{s} T_{Oi} / \sum_{i=1}^{s} T_{Ui}$
$prob_{SE}$	Probability of a selection error, computed as: $\sum_{i=1}^{s} N_{Oi} / \sum_{i=1}^{s} N_{Ti}$

4. SIMULATION RESULTS

We implemented our experiment as an SLX™ [28] simulation of Jini lookup servers, services, clients, and directors, executing a set of runs that each considered an increasing population of clients, each supported by a director using one of the selection algorithms defined in Section 3.4. Each client in each run generated 1,000 queries, and each run was iterated 100 times; thus, each data point observes c x 10^5 replica selections. Below, we report results in two sets: the four common selection algorithms and the two algorithms we invented.

Figures 3(a)-(c) plot performance (each graph displays a different metric) under increasing load for the common selection algorithms. Figures 4(a)-(c) plot performance for the balanced and balanced-partitioned algorithms, where we increase beyond 100 clients in order to distinguish performance differences. Figures 4(a)-(c) also include for comparison weighted selection, the best performing of the common algorithms.

5. DISCUSSION

Our results show that selecting replicas based on information yields superior performance over random selection, which may assign transactions to overloaded replicas; thus leading to higher response times and server latencies. One exception appears: the "thundering herd" effect induced by greedy selection causes higher variance in server latency (not shown), as transactions descend en masse upon the best performing replica, transforming it to a poor performer. Bulk arrivals ensure that information on which decisions were based becomes outdated quickly.

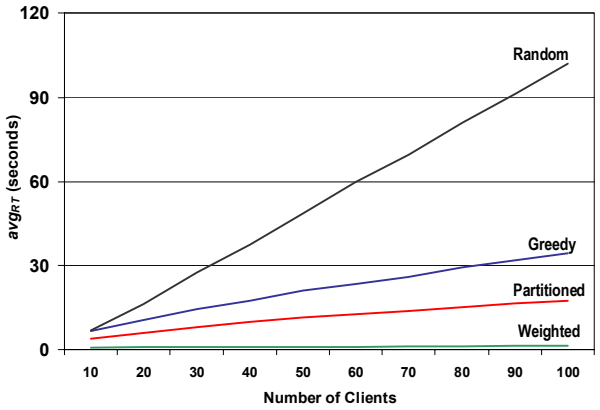

Figure 3(a). Average Client Response Time for Common Selection Algorithms

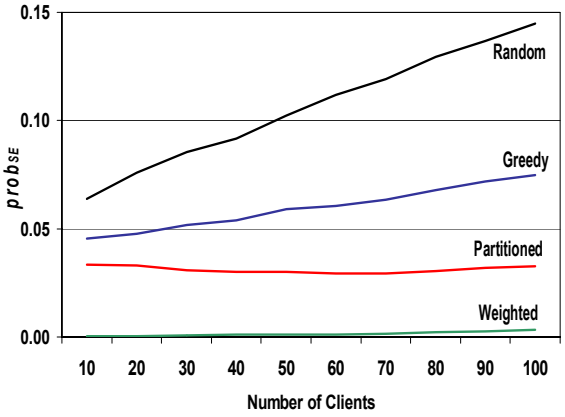

Figure 3(b). Probability of Selection Error for Common Selection Algorithms

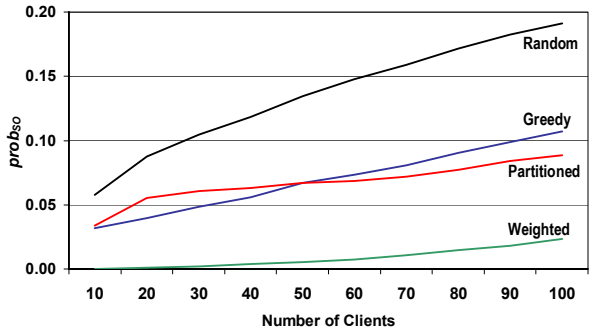

Figure 3(c). Probability of Server Overload for Common Selection Algorithms

Partitioning replicas into two sets (based on server latency) and then selecting randomly among the less loaded set, provides general improvement over greedy selection on all metrics. Further, the advantage of partitioned selection increases with client load. Spreading transactions evenly among replicas likely to provide good performance does not rapidly push one particular replica into overload. The general advantage of partitioned (random) over greedy selection exhibits one exception. Below 50 clients, servers have higher probability of being overloaded with partitioned (random) selection because the greedy algorithm assigns work in series – replica by replica – causing the number of overloaded servers to increase more slowly. Once all replicas reach saturation, the bulk arrival process of greedy selection creates larger backlogs, while partitioned selection spreads arrivals more evenly, allowing servers to spend less time in overload.

Among the common algorithms, weighted selection provides the best performance on all metrics. Weighted selection adapts to changes in replica state without inducing rapid or large fluctuations. Greedy selection stimulates large changes in workload, pushing a selected replica away from the state that led to its selection. The partitioned algorithm induces cyclic oscillation in replica workload, but at a somewhat slower frequency than greedy selection. Weighted selection tends mainly to react to changes in replica state, while the greedy and partitioned algorithms induce feedback that alters the state to which they are reacting. This difference leads weighted selection to exhibit more stable and desirable performance.

The balanced algorithm shares the reactive nature of weighted selection but improves performance for two reasons. First, balanced selection assigns more transactions to replicas with greater available processing capacity. Second, using the replica with the largest estimated server latency as the goal state reduces pressure for upward movement in system-wide server latency, and tends to reinforce downward movement. These reasons also explain why the balanced-partitioned algorithm performs well, up to a point. As the client population surpasses 100, performance degrades for the balanced-partitioned algorithm because the set of replicas available diminishes, forcing fewer replicas to receive more transactions. After load reaches saturation, partitioning creates a bulk-arrival process that pushes replicas into overload for longer periods. These results indicate that adding a partitioning step could diminish performance for an otherwise good selection algorithm.

6. CONCLUSIONS

We used simulation to characterize performance (response time, selection error, probability of server overload) for four common replica-selection algorithms (random, greedy, partitioned, weighted) when applied in a decentralized form to client queries in a distributed object system deployed on a local network. We introduced two new selection algorithms (balanced and balanced-partitioned) that give improved performance over the more common algorithms. We found that weighted selection performs best among the common algorithms and that balanced selection performs best overall. We explained why greedy and random algorithms should be avoided. We also provided evidence that preceding selection with a partitioning step can weaken an otherwise good selection algorithm.

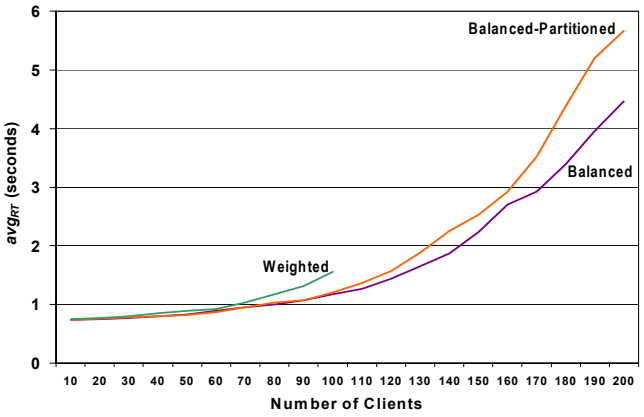

Figure 4(a). Average Client Response Time for New (and Weighted) Selection Algorithms

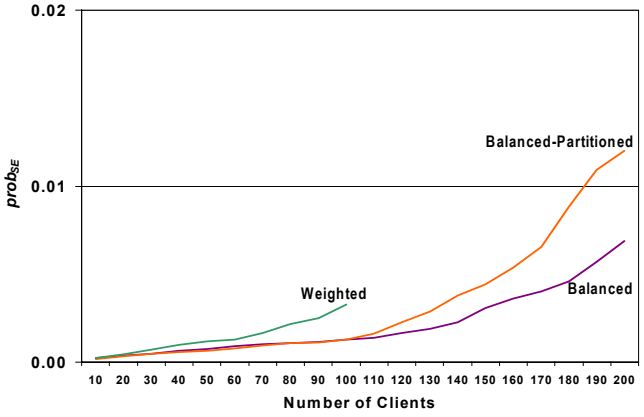

Figure 4(b). Probability of Selection Error for New (and Weighted) Selection Algorithms

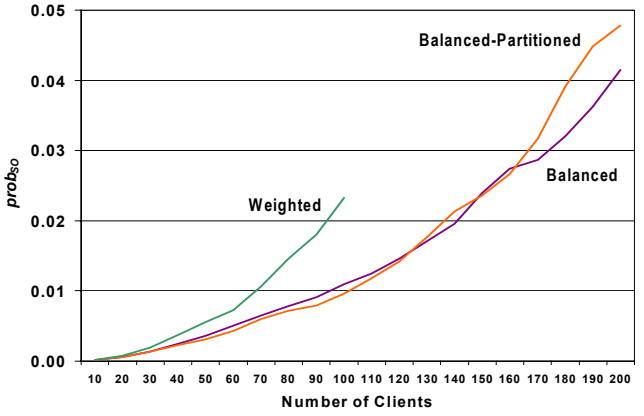

Figure 4(c). Probability of Server Overload for New (and Weighted) Selection Algorithms

7. REFERENCES

[1] Rabinovich, M., Xiao, Z., and Aggarwal, A. Computing on the Edge: A Platform for Replicating Internet Applications. In *Proceedings of the 8th International Workshop on Web Content Caching and Distribution*, (Hawthorne, New York, September 29 through October 1, 2003).

[2] Lewontin, S. and Martin, E. Client Side Load Balancing for the Web. In *Proceedings of 6th International World Wide Web Conference*. (Santa Clara, California, April 7-11, 1997).

[3] Vingralek, R., Breitbart, Y., Sayal, M., and Scheuermann, P. Web++: A System For Fast and Reliable Web Service. In *Proceedings of the USENIX Annual Technical Conference*. (Monterey, California, June 6-11, 1999). USENIX Association.

[4] Sayal, M., Scheuermann, P., and Vingralek, R. Content Replication in Web++. In *Proceedings 2nd IEEE International Symposium on Network Computing and Applications*. (Cambridge, Massachusetts, April 16 - 18, 2003). IEEE, p. 33.

[5] Fei, Z., Bhattacharjee, S., Zegura, E., and Ammar, M. A Novel Server Selection Technique for Improving Response Time of a Replicated Service. In *Proceedings IEEE INFOCOM 1998*. (San Francisco, California, March 1998). IEEE, pp. 783-791.

[6] Crovella, M. and Carter, R. Dynamic Server Selection in the Internet. In *Proceedings of the 3rd IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems*. (Mystic, Connecticut, August 1995).

[7] Carter, R. and Corvella, M. Server Selection using Dynamic Path Characterization in Wide-Area Networks. In *Proceedings of INFOCOM 1997*. (Kobe, Japan, April 1997).

[8] Cardellini, V., Colajanni, M. and Yu, P. Request Redirection Algorithms for Distributed Web Systems. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 14, No. 4, April 2003, pp. 355-368.

[9] Sayal, M., Breitbart, Y., Scheuermann, P. and Vingralek, R. Selection Algorithms for Replicated Web Servers. In *Proceedings of the Workshop on Internet Server Performance*. (Madison, Wisconsin, June 1998).

[10] Connect Control Datasheet. Check Point Software Technologies Ltd. 2003.

[11] Load Balancing System, Chapter 6 in Intel Solutions Manual, Intel Corporation, pp. 49-67.

[12] Farrell, R. Review of Web server load balancers. Network World, September 27, 1997.

[13] Load Balancing in a Cluster, WebLogic Server 7.0, bea.

[14] Configuring application server load balancing, Tarantella.

[15] Server Load Balancing. TechBrief from Extreme Networks.

[16] Othman, O., O'Ryan, C. and Schmidt, D. The Design and Performance of an Adaptive CORBA Load Balancing Service. To appear in the "online" edition of the *Distributed Systems Engineering Journal*. February 2001.

[17] Krishnamurthy, S. Sanders, W., and Cukier, M. Performance Evaluation of a Probabilistic Replica Selection Algorithm. *In Proceedings of the Seventh IEEE International Workshop on Object-Oriented Real-Time Dependable System*s. (San Diego, California January 07 - 09, 2002).

[18] Shen, K., Yang, T., and Chu, L. Cluster Load Balancing for Fine-Grained Network Services. In *Proceedings of the International Parallel and Distributed Processing Symposium* (IPDPS). (Fort Lauderdale, Florida April 15-19, 2002).

[19] Waldvogel, M., Hurley, P., and Bauer, D. Dynamic Replica Management in Distributed Hash Tables. IBM Research Report RZ-3502, July 2003.

[20] Ferdean, C. and Makpangou, M. A Scalable Replica Selection Strategy based on Flexible Contracts. In *Proceedings of the Third IEEE Workshop on Internet Applications*. (San Jose, California, June 23 - 24, 2003).

[21] Vazhkudai, S. Tuecke, S., and Foster, I. Replica Selection in the Globus Data Grid. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*. (Brisbane, Australia, May 15-18, 2001).

[22] Zhao, Y. and Hu, Y. GRESS – a Grid Replica Selection Service. In *Proceedings of the 15th International Conference Parallel And Distributed Computing and Systems*. (Marina Del Ray, California, November 3-5, 2003).

[23] Fu, Z. and Venkatasubramanian, N. Combined Path and Server Selection in Dynamic Multimedia Environments. In *Proceedings of the 7th ACM International Conference on Multimedia* (Part 1). (Orlando, Florida, 1999). ACM pp. 469-472.

[24] Guo, M. Ammar, M. Zegura, E. Selecting among Replicated Batching Video-on-Demand Servers. In *Proceedings of the 12th International Workshop on Network and Operating System Support for Digital Audio and Video*. (Miami, Florida, May 12-14, 2002).

[25] Huang, C. and Abdelzaher, T. Towards Content Distribution Networks with Latency Guarantees. In *Proceedings of the 12th International Workshop on Quality of Service*. (Montreal, Canada, June 7-9, 2004).

[26] Krishnamurthy, B. Wills, C. and Zhang, Y. On the Use and Performance of Content Distribution Networks. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*. (San Francisco, California, November 1-2, 2001).

[27] Arnold, K. et al, The Jini Specification, V1.0 Addison-Wesley 1999. Latest version is available from Sun.

[28] Henriksen, J. An Introduction to SLX™. In *Proceedings of the 1997 Winter Simulation Conference*. (Atlanta, Georgia, December 7-10, 1997), pp. 559-566.